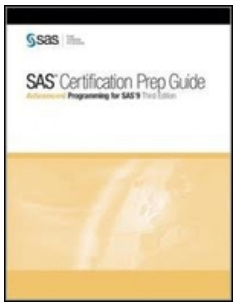


SAS Certification Prep Guide: Advanced Programming for SAS 9, Third Edition

by SAS Institute

Chapter 4: Combining Tables Vertically Using PROC SQL

Overview

Introduction

Suppose you are generating a report based on data from a health clinic. You want to display the results of individual patient stress tests taken in 1998 (which are stored in Table A), followed by the results from stress tests taken in 1999 (which are stored in Table B). Instead of combining the table rows horizontally, as you would in a PROC SQL Join, you want to combine the table rows *vertically* (one on top of the other).

Table A
Table B

When you need to select data from multiple tables and combine the tables vertically, PROC SQL can be an efficient alternative to using other SAS procedures or the DATA step. In a PROC SQL *set operation*, you use one of four *set operators* (EXCEPT, INTERSECT, UNION, and OUTER UNION) to combine tables (and views) vertically by combining the results of two queries:

```
proc sql;
   select *
      from a
   set-operator
   select *
      from b;
```

Each set operator combines the query results in a different way.

In this chapter, you will learn how to use the various set operators, with or without the optional keywords ALL and CORR (CORRESPONDING), to combine the results of multiple queries.

Note In this chapter, the references to tables are also applicable to views, unless otherwise noted.

Objectives

In this chapter, you learn to

- combine the results of multiple PROC SQL queries in different ways by using the set operators EXCEPT, INTERSECT, UNION, and OUTER UNION
- modify the results of a PROC SQL set operation by using the keywords ALL and CORR (CORRESPONDING)
- compare PROC SQL outer unions with other SAS techniques.

Prerequisites

Before beginning this chapter, you should complete the following chapters:

- "Performing Queries Using PROC SQL" on page 4
- "Performing Advanced Queries Using PROC SQL" on page 29
- "Combining Tables Horizontally Using PROC SQL" on page 86.

Understanding Set Operations

Overview

A set operation contains

- two queries (each beginning with a SELECT clause)

- a set operator
- one or both of the keywords ALL and CORR (CORRESPONDING).

General form of an SQL query using a set operator:

```
SELECT column-1<,  column-n>
     FROM table-1 | view-1<, … table-n | view-n>
     <optional query clauses>
set-operator <ALL> <CORR>
SELECT column-1<, … column-n>
     FROM table-1 | view-1<, … table-n | view-n>
     <optional query clauses>;
```

where

SELECT
specifies the column(s) that will appear in the result.

FROM
specifies the table(s) or view(s) to be queried.

optional query clauses
are used to refine the query further and include the clauses WHERE, GROUP BY, HAVING, and ORDER BY.

- the *set-operator* is one of the following: EXCEPT|INTERSECT|UNION|OUTER UNION.
- the optional keywords *ALL* and *CORR (CORRESPONDING)* further modify the set operation.

The query or set operation contains one semicolon, which is placed after the last SELECT statement.

Example

In the following PROC SQL step, the SELECT statement contains one set operation. The set operation uses the set operator UNION to combine the result of a query on the table *Sasuser.Stress98* with the result of a query on the table *Sasuser.Stress99*.

```
proc sql;
   select *
      from sasuser.stress98
   union
   select *
      from sasuser.stress99;
```

You will learn the details about using each set operator later in this chapter.

Processing a Single Set Operation

PROC SQL evaluates a SELECT statement with one set operation as follows:

a. Each query is evaluated to produce an intermediate (internal) result table.

b. Each intermediate result table then becomes an operand linked with a set operator to form an expression(for example, **`Table1` *UNION* `Table2`**).

c. PROC SQL evaluates the entire expression to produce a single output result set.

Using Multiple Set Operators

A single SELECT statement can contain more than one set operation. Each additional set operation includes a set operator and a group of query clauses, as shown in the following example:

```
proc sql;
```

```
select *
   from table1
set-operator
select *
   from table2
set-operator
select *
   from table3;
```

This SELECT statement uses two set operators to link together three queries.

Regardless of the number of set operations in a SELECT statement, the statement contains only one semicolon, which is placed after the last query.

Example

The following PROC SQL step contains two set operators (both are OUTER UNION) that combine three queries:

```
proc sql;
   select *
      from sasuser.mechanicslevel1
   outer union
   select *
      from sasuser.mechanicslevel2
   outer union
   select *
      from sasuser.mechanicslevel3;
```

Processing Multiple Set Operations

When PROC SQL evaluates a SELECT statement that contains multiple set operations, an additional processing step (step 3 below) is required:

1. Each query is evaluated to produce an intermediate (internal) result table.
2. Each intermediate result table then becomes an operand linked with a set operator to form an expression(for example, **`Table1 UNION Table2`**).
3. *If the set operation contains more than two queries*, then the result from the first two queries (enclosed in parentheses in the following examples) becomes an operand for the next set operator and operand. For example:
   - with two set operators: **`(Table1 UNION Table2) EXCEPT Table3`**
   - with three set operators: **`((Table1 UNION Table2) EXCEPT Table3) INTERSECT Table4`**.
4. PROC SQL evaluates the entire expression to produce a single output result set.

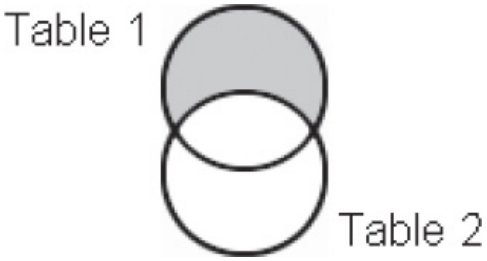
Note When processing set operators, PROC SQL follows a default order of precedence, unless this order is overridden by parentheses in the expression(s). By default, INTERSECT is evaluated first. OUTER UNION, UNION, and EXCEPT all have the same level of precedence.

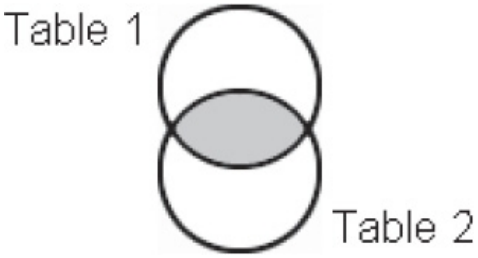
Introducing Set Operators

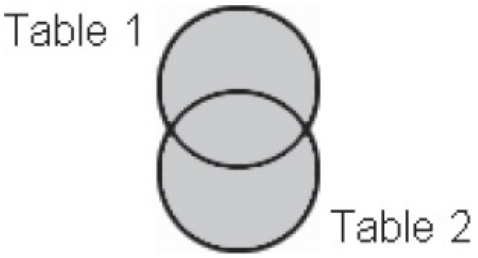
Each of the four set operators EXCEPT, INTERSECT, UNION, and OUTER UNION selects rows and handles columns in a different way, as described below.

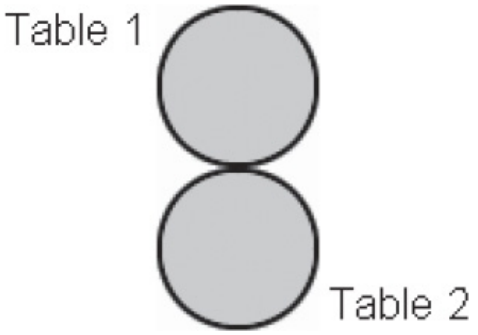
Note In the following chart, Table 1 is the table that is referenced in the first query and Table 2 is the table that is referenced in the second query.

Set Operator	Treatment of Rows	Treatment of Columns	Example
EXCEPT	Selects *unique* rows from *the first* table that are *not*	*Overlays* columns based on then	`proc sql;`

	found in the *second* table.
	 
		position in the SELECT clause without regard to the individual column names.	`select *`
			`from table1`
			`except`
			`select *`
			`from table2;`

INTERSECT	Selects *unique* rows that are *common* to *both* tables.
	 
		Overlays columns based on their *position* in the SELECT clause without from table1 regard to the individual intersect column names.	`proc sql;`
			`select *`
			`from table1`
			`intersect`
			`select *`
			`from table2;`

UNION	Selects *unique* rows from *both* tables.
	 
		Overlays columns based on *their position* in the SELECT clause without regard to the individual column names.	`proc sql;`
			`select *`
			`from table1`
			`union`
			`select *`
			`from table2;`

OUTER UNION	Selects *all* rows from *both* tables.
	The OUTER UNION operator *concatenates* the results of the queries.
	 
		Does *not* overlay columns.	`proc sql;`
			`select *`
			`from table1`
			`outer union`
			`select *`
			`from table2;`

Note A set operator that selects only *unique* rows will display *one occurrence* of a given row in output.

Processing Unique versus Duplicate Rows

When processing a set operation that displays *only unique rows* (a set operation that contains the set operator EXCEPT, INTERSECT, or UNION), PROC SQL makes *two passes* through the data, by default:

1. PROC SQL eliminates duplicate (nonunique) rows in the tables.
2. PROC SQL selects the rows that meet the criteria and, where requested, overlays columns.

For set operations that display *both unique and duplicate rows*, only *one pass* through the data (step 2 above) is required.

Combining and Overlaying Columns

You can use a set operation to combine tables that have different numbers of columns and rows or that have columns in a

different order.

Three of the four set operators (EXCEPT, INTERSECT, and UNION) combine columns by overlaying them. (The set operator OUTER UNION does not overlay columns.)

By default, the set operators EXCEPT, INTERSECT, and UNION overlay columns based on the relative *position* of the columns in the SELECT clause. Column names are ignored. You control how PROC SQL maps columns in one table to columns in another table by specifying the columns in the appropriate order in the SELECT clause. The first column specified in the first query's SELECT clause and the first column specified in the second query's SELECT clause are overlaid, and so on.

When columns are overlaid, PROC SQL uses the column name from the first table (the table referenced in the first query). If there is no column name in the first table, the column name from the second table is used. When the SELECT clause contains an asterisk (*) instead of a list of column names, the set operation combines the tables (and, if applicable, overlays columns) based on the positions of the columns in the tables.

For example, the following set operation uses the set operator EXCEPT, so columns are overlaid. The SELECT clause in each query uses an asterisk (*), so the columns are overlaid based on their positions in the tables. The first column in table *One* **(X)** is overlaid on the first column in table *Two* **(X)**, and so on.

```
proc sql;
   select *
      from one
   except
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X	A
1	a
1	b
2	c
4	e
6	g

In order to be overlaid, columns in the same relative position in the two SELECT clauses must have the *same data type*. If they do not, PROC SQL generates a warning message in the SAS log and stops executing. For example, in the tables shown above, if the column **`One.X`** had a different data type than column **`Two.X`**, the SAS log would display the following error message.

Table 4.1: SAS Log

```
ERROR: Column 1 from the first contributor of EXCEPT
is not the same type as its counterpart from the second.
```

Next, we will use the keywords ALL and CORR to modify the default action of the set operators.

Modifying Results by Using Keywords

To modify the behavior of set operators, you can use either or both of the keywords *ALL* and *CORR* immediately following the set operator:

```
proc sql;
   select *
      from table1
   set-operator <all> <corr>
   select *
      from table2;
```

The use of each keyword is described below.

Keyword	Action	Used When…
ALL	Makes only *one pass* through the data and does *not* remove duplicate rows.	You do not care if there are duplicates. Duplicates are not possible. *ALL cannot* be used with OUTER UNION.
CORR (or *CORRESPONDING*)	*Compares* and overlays columns by *name* instead of by position: ▪ When used with *EXCEPT, INTERSECT*, and *UNION, removes* any columns that do not have the same name in both tables. ▪ When used with *OUTER UNION, overlays* same-named columns and displays columns that have nonmatching names *without overlaying*. If an alias is assigned to a column in the SELECT clause, CORR will use the alias instead of the permanent column name.	Two tables have some or all columns in common, but the columns are not in the same order.

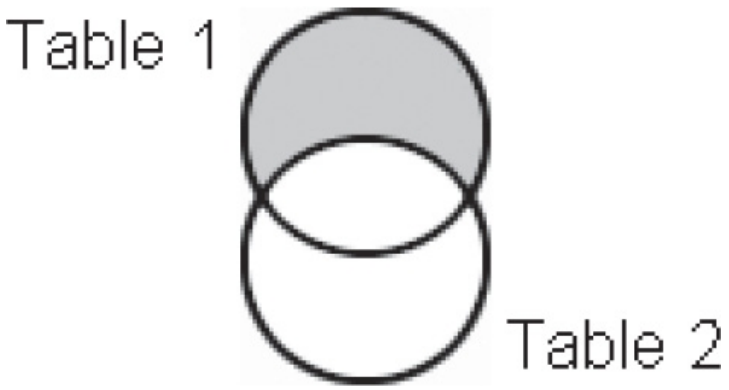
In the remainder of this chapter, you will learn more about the use of each set operator, with and without the keywords ALL and CORR.

Using the EXCEPT Set Operator

Overview

The set operator EXCEPT does both of the following:

- selects *unique rows* from the *first* table (the table specified in the first query) that are *not found* in the *second* table (the table specified in the second query)
- overlays columns.



Consider how EXCEPT works when used alone and with the keywords ALL and CORR.

Using the EXCEPT Operator Alone

Suppose you want to display the *unique* rows in table *One* that are *not* found in table *Two*. The PROC SQL set operation that includes the EXCEPT operator, the tables *One* and *Two*, and the output of the set operation are shown below:

```
proc sql;
   select *
      from one
   except
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X	A
1	a
1	b
2	c
4	e
6	g

The set operator EXCEPT overlays columns by their *position*. In this output, the following columns are overlaid:

- the first columns, **`One.X`** and **`Two.X`**, both of which are numeric
- the second columns, **`One.A`** and **`Two.B`**, both of which are character.

The column names from table *One* are used, so the second column of output is named **A** rather than **B**.

Consider how PROC SQL selects rows from table *One* to display in output.

In the first pass, PROC SQL eliminates any *duplicate rows* from the tables. As shown below, there is one duplicate row: in table *One*, the second row is a duplicate of the first row. All remaining rows in table *One* are still candidates in PROC SQL's selection process.

```
proc sql;
   select *
      from one
   except
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

In the second pass, PROC SQL identifies any rows in table *One* for which there is a *matching row* in table *Two* and eliminates them. There is one matching row in the two tables, as shown below, which is eliminated.

```
proc sql;
   select *
      from one
   except
   select *
      from two;
```

One

X	A
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

The five remaining rows in table *One*, the unique rows, are displayed in the output.

X	A
1	a
1	b
2	c
4	e
6	g

Using the Keyword ALL with the EXCEPT Operator

To select *all rows* in the *first* table (both unique and duplicate) that do *not* have a matching row in the *second* table, add the keyword *ALL* after the EXCEPT set operator. The modified PROC SQL set operation, the tables *One* and *Two*, and the output are shown below:

```
proc sql;
   select *
      from one
   except all
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X	A
1	a
1	a
1	b
2	c
4	e
6	g

The output now contains six rows. PROC SQL has again eliminated the one row in table *One* (the fifth row) that has a matching row in table *Two* (the fourth row). Remember that when the keyword ALL is used with the EXCEPT operator, PROC SQL does *not* make an extra pass through the data to remove duplicate rows within table *One*.

Therefore, the second row in table *One*, which is a duplicate of the first row, is now included in the output.

Using the Keyword CORR with the EXCEPT Operator

To display both of the following, add the keyword *CORR* after the set operator.

- only columns that have the *same name*
- all *unique rows* in the *first* table that do *not* appear in the *second* table.

The modified PROC SQL set operation, the tables *One* and *Two*, and the output are shown below:

```
proc sql;
   select *
      from one
   except corr
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X
4
6

---

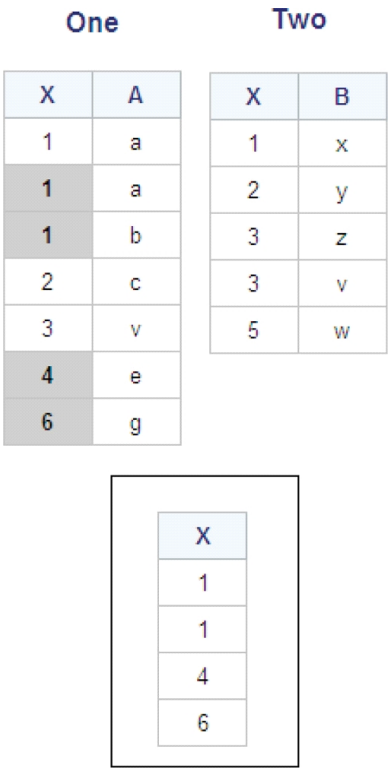
x is the only column that has the same name in both tables, so **x** is the only column that PROC SQL examines and displays in the output.

In the first pass, PROC SQL eliminates the second and third rows of table *One* from the output because they are not unique within the table; they contain values of **x** that duplicate the value of **x** in the first row of table *One*. In the second pass, PROC SQL eliminates the first, fourth, and fifth rows of table *One* because each contains a value of **x** that matches a value of **x** in a row of table *Two*. The output displays the two remaining rows in table *One*, the rows that are unique in table *One* and that do *not* have a row in table *Two* that has a matching value of **x**.

Using the Keywords ALL and CORR with the EXCEPT Operator

If the keywords ALL and CORR are used together, the EXCEPT operator will display all *unique and duplicate* rows in the *first* table that do *not* appear in the *second* table, and will overlay and display only columns that have the *same name*. The modified PROC SQL set operation, the tables *One* and *Two*, and the output are shown below:

```
proc sql;
   select *
      from one
   except all corr
   select *
      from two;
```



One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X
1
1
4
6

Once again, PROC SQL looks at and displays only the column that has the same name in the two tables: **x**. Because the ALL keyword is used, PROC SQL does *not* eliminate any duplicate rows in table *One*. Therefore, the second and third rows in table *One*, which are duplicates of the first row in table *One*, appear in the output. PROC SQL does eliminate the first, fourth, and fifth rows in table *One* from the output because for each one of these three rows there is a corresponding row in table *Two* that has a matching value of **x**.

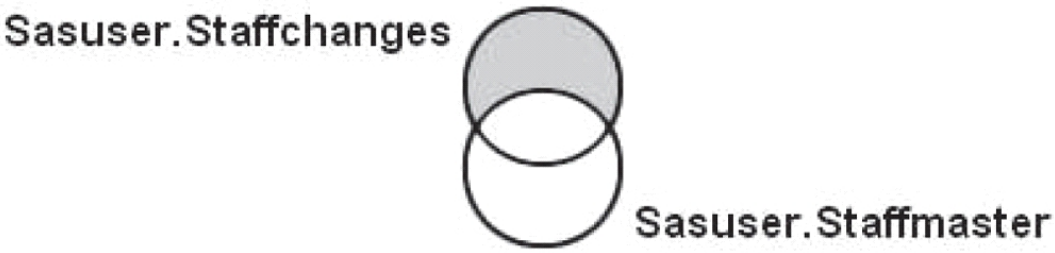
As this example shows, when the ALL keyword is used with the EXCEPT operator, a row in table *One* cannot be eliminated from the output unless it has a *separate* matching row in table *Two*. Table *One* contains three rows in which the value of **x** is *1*, but table *Two* contains only one row in which the value of **x** is *1*. That one row in table *Two* causes the first of the three rows in table *One* that have a matching value of **x** to be eliminated from the output. However, table *Two* does *not* have two *additional* rows in which the value of **x** is *1*, so the other two rows in table *One* are not eliminated, and do appear in the output.

Example: EXCEPT Operator

The EXCEPT operator can be used to solve a realistic business problem. Suppose you want to display the names of all *new* employees of a company. There is no table that contains information for only the new employees, so you will have to use data from the following two tables.

Table	Relevant Columns
Sasuser.Staffchanges lists information for all new employees and existing employees who have had a change in salary or job code	`FirstName, LastName`
Sasuser.Staffmaster lists information for all existing employees	`FirstName, LastName`

The relationship between these two tables is shown in the diagram below:



The intersection of these two tables includes information for all existing employees who have had changes in job code or salary. The shaded portion, the portion of *Sasuser.Staffchanges* that does *not* overlap with *Sasuser.Staffmaster*, includes information for the people that you want: new employees.

To separate the new employees from the existing employees in *Sasuser.Staffchanges*, you create a set operation that displays all rows from the first table *(Sasuser.Staffchanges)* that do *not* exist in the second table *(Sasuser.Staffmaster)*. The following PROC SQL step solves the problem:

```
proc sql;
   select firstname, lastname
      from sasuser.staffchanges
   except all
   select firstname, lastname
      from sasuser.staffmaster;
```

This PROC SQL set operation includes the operator *EXCEPT* and the keyword *ALL*. Although you do not want the output to contain duplicate rows, you already know that there are no duplicates in these two tables. Therefore, ALL is specified to prevent PROC SQL from making an extra pass through the data, which speeds up the processing of this query.

PROC SQL compares only the columns that are specified in the SELECT clauses, and these columns are compared in the order in which they are specified. The output displays the first and last names of the two new employees.

FirstName	LastName
AMY	BRIDESTON
JIM	POWELL

Note In a set operation that uses the EXCEPT operator, the order in which the tables are listed in the SELECT statement makes a difference. If the tables in this example were listed in the opposite order, the output would display all *existing* employees who have had *no changes* in salary or job code.

Example: EXCEPT Operator in an In-Line View

This example is a variation of the preceding set operation. Suppose you want to display the *number of existing employees* who have had *no changes* in salary or job code. Once again, the query uses the following tables and columns.

Table	Relevant Columns
Sasuser.Staffchanges lists information for all new employees and existing employees who have had a change in salary or job code	`FirstName, LastName`
Sasuser.Staffmaster lists information for all existing employees	`FirstName, LastName`

The following PROC SQL query solves this problem:

```
proc sql;
   select count(*) label='No. of Persons'
      from (select EmpID
              from sasuser.staffmaster
            except all
            select EmpID
              from sasuser.staffchanges);
```

This PROC SQL query uses

- the COUNT function with an asterisk (*) as an argument to count the number of employee IDs returned from the set operation
- the set operator EXCEPT within an in-line view.

The in-line view returns a virtual table that contains employees who have had no changes in salary or job code. This virtual table is then passed to the COUNT(*) summary function, which counts the number of rows in the virtual table. The output shows that there are 144 existing employees who have had no changes in salary or job code.

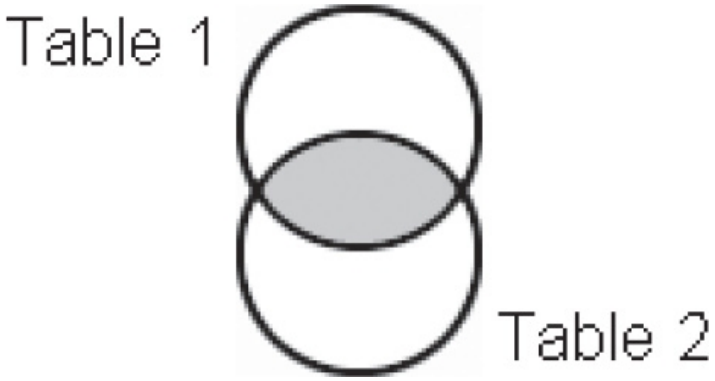
No.of Persons
144

Using the INTERSECT Set Operator

Overview

The set operator INTERSECT does both of the following:

- selects *unique rows* that are *common* to *both* tables
- overlays columns.



The following example demonstrates how INTERSECT works when used alone and with the keywords ALL and CORR.

Using the INTERSECT Operator Alone

The INTERSECT operator compares and overlays *columns* in the *same* way as the EXCEPT operator, by column position instead of column name. However, INTERSECT selects rows differently and displays in output the *unique rows* that are *common to both tables*. The following PROC SQL set operation uses the INTERSECT operator to combine the tables *One* and *Two*, which were introduced previously:

```
proc sql;
   select *
      from one
   intersect
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

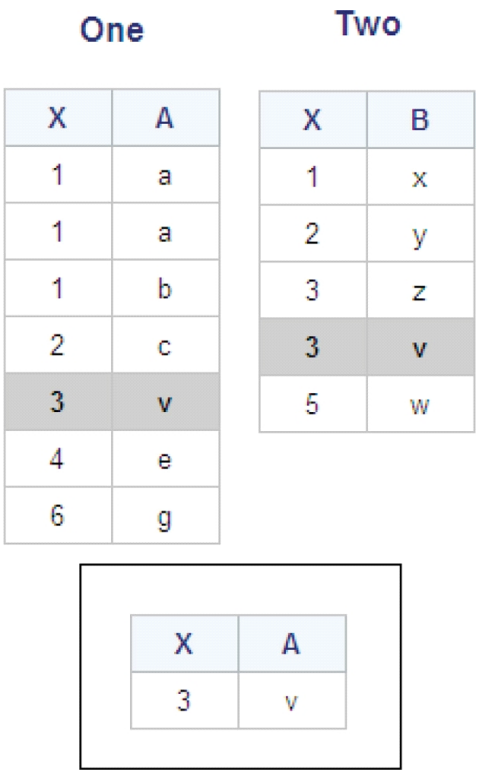
X	A
3	v

Tables *One* and *Two* have only one unique row in common and this row is displayed in the output. (This is the same row that was eliminated in the earlier example that contained the EXCEPT operator.)

Using the Keyword ALL with the INTERSECT Operator

Adding the keyword *ALL* to the preceding PROC SQL query prevents PROC SQL from making an extra pass through the data. If there were any rows common to tables *One* and *Two* that were duplicates of other common rows, they would also be included in output. However, as you have seen, there is only one common row in these tables. The modified PROC SQL query, the tables *One* and *Two*, and the output are shown below:

```
proc sql;
   select *
      from one
   intersect all
   select *
      from two;
```

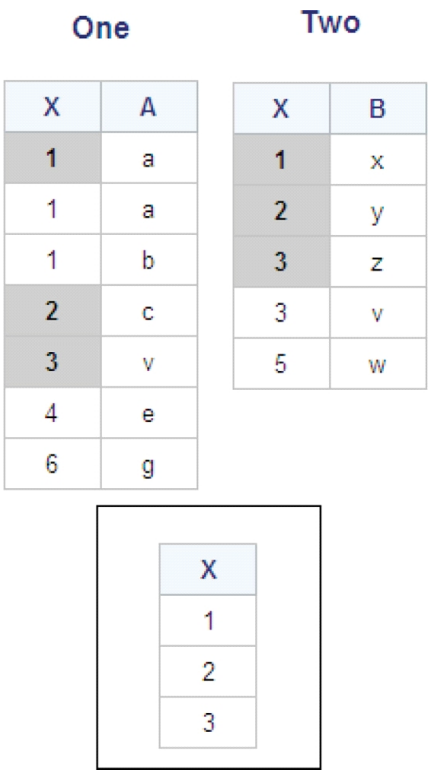


As before, there is just one row of output.

Using the Keyword CORR with the INTERSECT Operator

To display the unique rows that are common to the two tables based on the column *name* instead of the column position, add the *CORR* keyword to the PROC SQL set operation. The modified query, the tables *One* and *Two*, and the output are shown below:

```
proc sql;
   select *
      from one
   intersect corr
   select *
      from two;
```



One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X
1
2
3

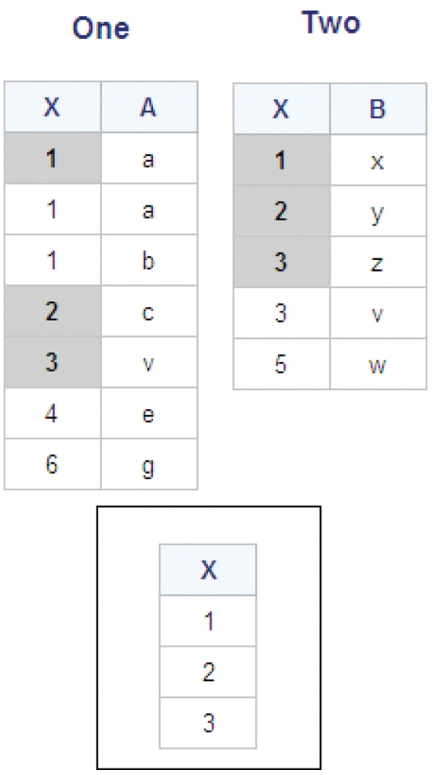
---

x is the only column name that is common to both tables, so **x** is the only column that PROC SQL examines and displays in the output. In the first pass, PROC SQL eliminates the rows that are duplicated within each table: the second and third rows in table *One* contain the same value for **x** as the first row, and the fourth row in table *Two* contains the same value for **x** as the third row. In the second pass, PROC SQL eliminates any rows that are not common across tables: the fourth and fifth rows in table *One* and the fifth row in table *Two* do not have a matching value of **x** in the other table. The output displays the three rows with *unique* values of **x** that are also *common* to both tables.

Using the Keywords ALL and CORR with the INTERSECT Operator

If the keywords ALL and CORR are used together, the INTERSECT operator will display all *unique and nonunique* (duplicate) rows that are common to the two tables, based on columns that have the same name. The modified query, the tables *One* and *Two*, and the output are shown below:

```
proc sql;
   select *
      from one
   intersect all corr
   select *
      from two;
```



PROC SQL examines and displays only the column with the same name, **x**. There are three common rows across the two tables, which are highlighted above, and these are the three rows that are displayed in the output.

Note that each of the tables contains at least one other row that duplicates a value of **x** in one of the common rows. For example, in the second and third rows in table *One*, the value of **x** is *1*, as in one of the common rows. However, in order to be considered a common row and to be included in the output, every duplicate row in one table must have a *separate* duplicate row in the other table. In this example, there are no rows that have duplicate values and that are also common across tables. Therefore, in this example, the set operation with the keywords ALL and CORR generates the same output as with the keyword CORR alone.

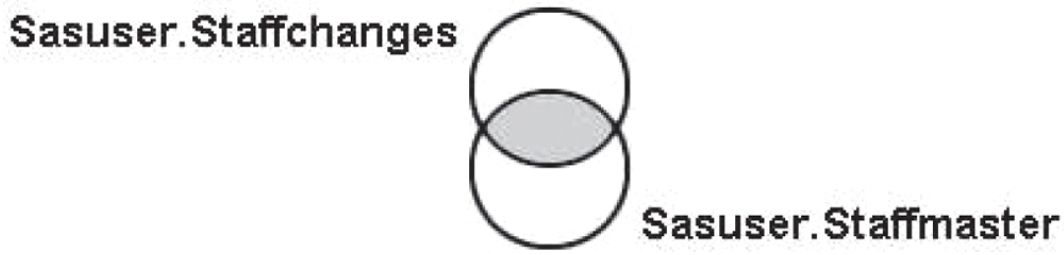
Example: INTERSECT Operator

Now that you have seen how the INTERSECT set operator works with very small tables, we can use INTERSECT in a realistic business problem. Suppose you want to display the names of the existing employees who *have changed* their salary or job code. (This query is the opposite of the query that you solved with the EXCEPT operator.)

Once again, you will use the following tables.

Table	Relevant Columns
Sasuser.Staffchanges lists information for all new employees and existing employees who have had a change in salary or job code	`FirstName, LastName`
Sasuser.Staffmaster lists information for all existing employees	`FirstName, LastName`

The relationship between these two tables is shown in the diagram below:



As shown in the earlier example with EXCEPT, the intersection of these two tables includes information for all existing employees who have had changes in job code or salary. It is the intersection of these two tables, shaded above, that you want to display.

To display the unique rows that are common to both tables, you use a PROC SQL set operation that contains INTERSECT. It is known that these tables contain no duplicates, so ALL is used to speed up query processing. The PROC SQL set operation is shown below:

```
proc sql;
   select firstname, lastname
      from sasuser.staffchanges
   intersect all
   select firstname, lastname
      from sasuser.staffmaster;
```

Note In this PROC SQL step, which contains just *one* INTERSECT set operator, the order in which you list the tables in the SELECT statement does *not* make a difference. However, in a more complex PROC SQL step that contains multiple stacked INTERSECT set operators, it is important to think through the table order carefully, depending on when you want the non-matches to be eliminated. The output shows that there are four existing employees who have changed their salary or job code.

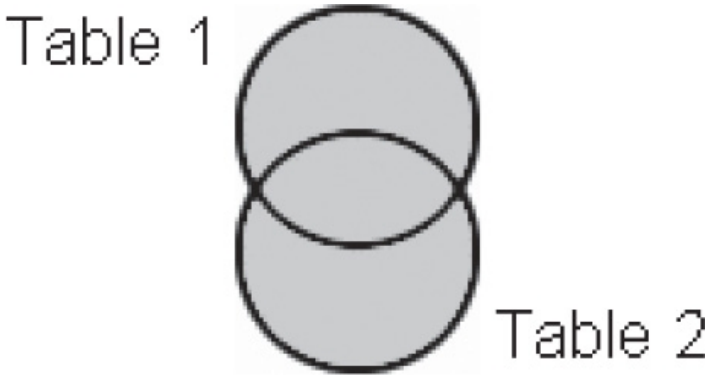
FirstName	LastName
DIANE	WALTERS
KAREN	CARTER
NEIL	CHAPMAN
RAYMOND	SANDERS

Using the UNION Set Operator

Overview

The set operator UNION does both of the following:

- selects *unique rows* from both tables
- overlays columns.



The following example demonstrates how UNION works when used alone and with the keywords ALL and CORR.

Using the UNION Operator Alone

To display all rows from the tables *One* and *Two* that are *unique* in the combined set of rows from both tables, use a PROC SQL set operation that includes the UNION operator:

```
proc sql;
   select *
      from one
   union
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X	A
1	a
1	b
1	x
2	c
2	y
3	v
3	z
4	e
5	w
6	g

With the UNION operator, PROC SQL first concatenates and sorts the rows from the two tables, and eliminates any duplicate rows. In this example, two rows are eliminated: the second row in table *One* is a duplicate of the first row, and the fourth row in table *Two* matches the fifth row in table *One*. All remaining rows, the unique rows, are included in the output. The columns are overlaid by position.

Using the Keyword ALL with the UNION Operator

When the keyword ALL is added to the UNION operator, the output displays *all* rows from both tables, both *unique and duplicate*. The modified PROC SQL set operation, the tables *One* and *Two*, and the new output are shown below:

```
proc sql;
   select *
      from one
   union all
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g
1	x
2	y
3	z
3	v
5	w

When the ALL keyword is used, PROC SQL does not remove duplicates or sort the rows. The output now includes the two duplicate rows that were eliminated in the previous example: the second row in table *One* and the fourth row in table *Two*. Note that the rows are in a different order in this output than they were in the output from the previous set operation.

Using the Keyword CORR with the UNION Operator

To display all rows from the tables *One* and *Two* that are *unique* in the combined set of rows from both tables, based on columns that have the same *name* rather than the same position, add the keyword CORR after the set operator. The modified query, the tables *One* and *Two*, and the output are shown below:

```
proc sql;
   select *
      from one
   union corr
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X
1
2
3
4
5
6

---

x is the only column name that is common to both tables, so **x** is the only column that PROC SQL examines and displays in the output. In the combined set of rows from the two tables, there are duplicates of the values *1*, *2*, and *3*, and these duplicate rows are eliminated from the output. The output displays the six unique values of **x**.

Using the Keywords ALL and CORR with the UNION Operator

If the keywords ALL and CORR are used together, the UNION operator will display *all* rows in the two tables both *unique and duplicate*, based on the columns that have the same *name*. In this example, the output displays *all* 12 values for **x**, the one column that has the same name in both tables.

```
proc sql;
   select *
      from one
   union all corr
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X
1
1
1
2
3
4
6
1
2
3
3
5

Example: UNION Operator

The UNION operator can be used to solve a realistic business problem. Suppose you are generating a report based on data from a health clinic. You want to display the results of individual patient stress tests taken in 1998, followed by the results from stress tests taken in 1999. To do this, you will use the UNION operator to combine the tables *Sasuser.Stress98* and *Sasuser.Stress99*. These two tables are similar in structure:

- both tables contain nine columns that have the same names
- each row contains data for an individual patient.

You are not sure whether the tables contain duplicate records, but you do not want duplicates in your output. Because the tables have the same column structure, you can overlay the columns by position, and the CORR keyword is not necessary. The PROC SQL set operation and output are shown below (the rows are ordered by IDs.):

```
proc sql;
   select *
      from sasuser.stress98
   union
   select *
      from sasuser.stress99;
```

ID	Name	RestHR	MaxHR	RecHR	TimeMin	TimeSec	Tolerance	Year
2458	Murray, W	72	185	128	12	38	D	1998
2462	Aimers, C	68	171	133	10	5	I	1998
2501	Bonaventure,	78	177	139	11	13	I	1999
2523	Johnson, R	69	162	114	9	42	S	1998
2539	LaMance, K	75	168	141	11	46	D	1998
2544	Jones, M	79	187	136	12	26	N	1999
2552	Reberson, P	69	158	139	15	41	D	1999

2555	King, E	70	167	122	13	13	I	1998
2563	Pitts, D	71	159	116	10	22	S	1998
2568	Eberhardt, S	72	182	122	16	49	N	1999
2571	Nunnelly, A	65	181	141	15	2	I	1999
2572	Oberon, M	74	177	138	12	11	D	1998
2574	Peterson, V	80	164	137	14	9	D	1998
2575	Quigley, M	74	152	113	11	26	I	1998
2578	Cameron, L	75	158	108	14	27	I	1999
2579	Underwood, K	72	165	127	13	19	S	1999
2584	Takahashi, Y	76	163	135	16	7	D	1998
2586	Derber, B	68	176	119	17	35	N	1998
2588	Ivan, H	70	182	126	15	41	N	1999
2589	Wilcox, E	78	189	138	14	57	I	1998
2595	Warren, C	77	170	136	12	10	S	1999

Tip If you can determine that these tables have no duplicate records, you could add the keyword ALL to speed up processing by avoiding an extra pass through the data.

Example: UNION Operator and Summary Functions

We can demonstrate another realistic business problem, to see how summary functions can be used with a set operator (in this case, UNION). Suppose you want to display the following summarized data for members of a frequent-flyer program: total points earned, total points used, and total miles traveled. All three values can be calculated from columns in the table *Sasuser.Frequentflyers* by using summary functions.

You might wonder why set operations are needed when only one table is involved. If you wanted to display the three summarized values *horizontally*, in three separate columns, you could solve the problem *without* a set operation, using the following simple SELECT statement:

```
proc sql;
   select sum(pointsearned) format=comma12.
          label='Total Points Earned',
          sum(pointsused) format=comma12.
          label='Total Points Used',
          sum(milestraveled) format=comma12.
          label='Total Miles Traveled'
   from sasuser.frequentflyers;
```

Total Points Earned	Total Points Used	Total Miles Traveled
10,583,453	4,429,670	10,477,963

Assume, however, that you want the three values to be displayed *vertically* in a single column. To generate this output, you create three different queries on the same table, and then use two UNION set operators to combine the three query results:

```
proc sql;
title 'Points and Miles Traveled';
title2 'by Frequent Flyers';
   select 'Total Points Traveled:',
          sum(MilesTraveled) format=comma12.
      from sasuser.frequentflyers
   union
   select 'Total Points Earned:',
          sum(PointsEarned) format=comma12.
      from sasuser.frequentflyers
   union
   select 'Total Points Used:',
          sum(PointsUsed) format=comma12.
```

```
    from sasuser.frequentflyers;
```

Each SELECT clause defines two columns: a character constant as a label and the summarized value. The output is shown below.

Points and Miles Traveled by Frequent Flyers	
Total Points Earned:	10,583,463
Total Points Traveled:	10,477,963
Total Points Used:	4,429,670

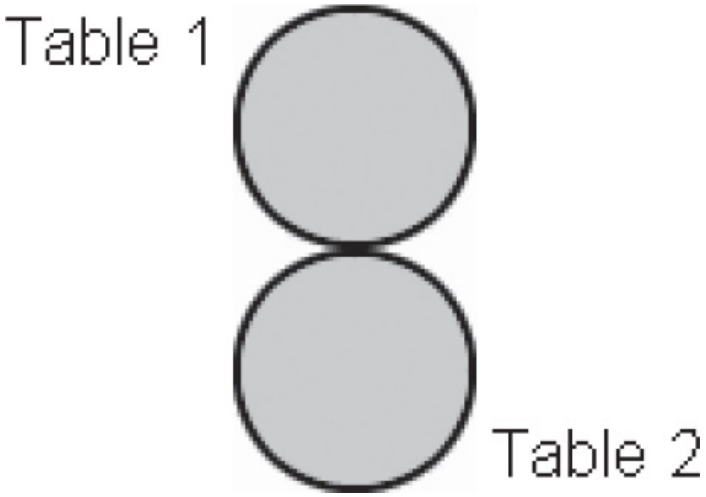
Note The preceding program reads the same table three times, so it is *not* the most efficient way to solve this problem.

Using the OUTER UNION Set Operator

Overview

The set operator OUTER UNION *concatenates* the results of the queries by

- *selecting all rows* (both unique and nonunique) from *both* tables
- *not overlaying* columns.



We can demonstrate how OUTER UNION works when used alone and with the keyword CORR. The ALL keyword is not used with OUTER UNION because this operator's default action is to include all rows in output.

Using the OUTER UNION Operator Alone

Suppose you want to display *all* rows from *both* of the tables *One* and *Two, without overlaying* columns. The PROC SQL set operation that includes the OUTER UNION operator, the two tables, and the output are shown below:

```
proc sql;
   select *
      from one
   outer union
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X	A	X	B
1	a	.	
1	a	.	
1	b	.	
2	c	.	
3	v	.	
4	e	.	
6	g	.	
.		1	x
.		2	y
.		3	z
.		3	v
.		5	w

In the output, the columns have *not* been overlaid. Instead, all four columns from both tables are displayed. Each row of output contains missing values in the two columns that correspond to the other table.

Using the Keyword CORR with the OUTER UNION Operator

The output from the preceding set operation contains two columns with the same name. To *overlay* the columns with a common name, add the CORR keyword to the set operation:

```
proc sql;
   select *
      from one
   outer union corr
   select *
      from two;
```

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X	A	B
1	a	
1	a	
1	b	
2	c	
3	v	
4	e	
6	g	
1		x
2		y
3		z
3		v
5		w

The output from the modified set operation contains only three columns, because the two columns named **x** are overlaid.

Example: OUTER UNION Operator

There are many business situations that require two or more tables to be concatenated. For example, suppose you want to display the employee numbers, job codes, and salaries of all mechanics working for an airline. The mechanic job has three levels and there is a separate table containing data for the mechanics at each level: *Sasuser.Mechanicslevel1*, *Sasuser.Mechanicslevel2*, and *Sasuser.Mechanicslevel3*. These tables all contain the same three columns.

The following PROC SQL step uses two OUTER UNION operators to concatenate the tables, and the CORR keyword to overlay the columns that have common names:

```
proc sql;
   select *
      from sasuser.mechanicslevel1
   outer union corr
   select *
      from sasuser.mechanicslevel2
   outer union corr
   select *
      from sasuser.mechanicslevel3;
```

EmpID	JobCode	Salary
1400	ME1	$41,677
1403	ME1	$39,301
1120	ME1	$40,067
1121	ME1	$40,757
1412	ME1	$38,919
1200	ME1	$38,942
1995	ME1	$40,334

1418	ME1	$39,207
1653	ME2	$49,151
1782	ME2	$49,483
1244	ME2	$51,695
1065	ME2	$49,126
1129	ME2	$48,901
1406	ME2	$49,259
1356	ME2	$51,617
1292	ME2	$51,367
1440	ME2	$50,060
1900	ME2	$49,147
1423	ME2	$50,082
1432	ME2	$49,458
1050	ME2	$49,234
1105	ME2	$48,727
1499	ME3	$60,235
1409	ME3	$58,171
1379	ME3	$59,170
1521	ME3	$58,136
1385	ME3	$61,460
1420	ME3	$60,299
1882	ME3	$58,153

Comparing Outer Unions and Other SAS Techniques

A PROC SQL set operation that uses the OUTER UNION operator is just one SAS technique that you can use to concatenate tables, as shown in the following programs. Program 1 is the PROC SQL set operation that was shown earlier in this chapter. Program 2 uses a different SAS technique to concatenate the hypothetical tables *One* and *Two*.

Program 1: PROC SQL OUTER UNION Set Operation with CORR

```
proc sql;
   create table three as
      select * from one
      outer union corr
      select * from two;
quit;
```

Program 2: DATA Step, SET Statement, and PROC PRINT Step

```
data three;
   set one two;
run;
proc print data=three noobs;
run;
```

These two programs create the same table as output, as shown below.

One

X	A
1	a
1	a
1	b
2	c
3	v
4	e
6	g

Two

X	B
1	x
2	y
3	z
3	v
5	w

X	A	B
1	a	
1	a	
1	b	
2	c	
3	v	
4	e	
6	g	
1		x
2		y
3		z
3		v
5		w

When tables have a same-named column, the PROC SQL outer union will *not* produce the same output unless the keyword CORR is also used. CORR causes the same-named columns (in this example, the two columns named **x)** to be overlaid; without CORR, the OUTER UNION operator will include both of the same-named columns in the result set. The DATA step program will generate only one column **x**.

The two concatenation techniques shown above also vary in efficiency. A PROC SQL set operation generally requires more computer resources but might be more convenient and flexible than the DATA step equivalent.

Summary

Contents

This section contains the following topics.

Text Summary

Understanding Set Operations

A set operation combines tables or views vertically (one on top of the other) by combining the results of two queries. A set operation is a SELECT statement that contains

- two queries (each beginning with a SELECT clause)
- one of the set operators EXCEPT, INTERSECT, UNION, and OUTER UNION
- one or both of the keywords ALL and CORR (CORRESPONDING) as modifiers.

A single SELECT statement can contain multiple set operations.

When processing a set operation that displays only unique rows (a set operation that contains the set operator EXCEPT, INTERSECT, or UNION), PROC SQL makes two passes through the data, by default. For set operations that display both unique and duplicate rows, only one pass through the data is required.

For the set operators EXCEPT, INTERSECT, and UNION, columns are overlaid based on the relative position of the columns in the SELECT clause rather than by column name. In order to be overlaid, columns in the same relative position in the two SELECT clauses must have the same data type.

One or both keywords can be used to modify the default action of a set operator.

Using the EXCEPT Set Operator

The set operator EXCEPT selects unique rows from the first table (the table specified in the first query) that are not found in the second table (the table specified in the second query) and overlays columns. This set operation can be modified by using either or both of the keywords ALL and CORR.

Using the INTERSECT Set Operator

The set operator INTERSECT selects unique rows that are common to both tables and overlays columns. This set operation can be modified by using either or both of the keywords ALL and CORR.

Using the UNION Set Operator

The set operator UNION selects unique rows from both tables and overlays columns. This set operation can be modified by using either or both of the keywords ALL and CORR.

Using the OUTER UNION Set Operator

The set operator OUTER UNION concatenates the results of two queries by selecting all rows (both unique and nonunique) from both tables and not overlaying columns. This set operation can be modified by using the keyword CORR.

Comparing Outer Unions and Other SAS Techniques

A PROC SQL set operation that uses the OUTER UNION set operator is not the only way to concatenate tables in SAS. Other SAS techniques can be used, such as a program that consists of a DATA step, a SET statement, and a PROC PRINT step.

Syntax

```
PROC SQL;
     SELECT column-1<,… column n>
         FROM table-1 | view-1<, … table-n | view-n>
         <optional query clauses>
     set-operator <ALL> <CORR>
     SELECT column-1<,… column-n>
         FROM table-1 | view-1<, … table-n | view-n>
         <optional query clauses>",
QUIT;
```

Sample Program

```
proc sql;
   select firstname, lastname
      from sasuser.staffchanges
   intersect all
   select firstname, lastname
      from sasuser.staffmaster;
quit;
```

Points to Remember

- Regardless of the number of set operations in a SELECT statement, the statement contains only one semicolon, which is placed after the last query.
- In order to be overlaid, columns must have the same data type.

Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. Which statement is *false* with respect to a set operation that uses the EXCEPT, UNION, or INTERSECT set operator without a keyword? ?

a. Column names in the result set are determined by the first table.

b. To be overlaid, columns must be of the same data type.

c. To be overlaid, columns must have the same name.

d. By default, only unique rows are displayed in the result set.

2. The keyword ALL *cannot* be used with which of the following set operators? ?

a. EXCEPT

b. INTERSECT

c. UNION

d. OUTER UNION

3. Which PROC SQL step combines the tables *Summer* and *Winter* to produce the output displayed below? ?

Summer

Month	Temp	Precip
7	78	.05
8	85	.04
9	83	.15

Winter

Mo	Temp	Precip
1	29	.15
2	32	.17
3	38	.20
2	32	.17

Month	Temp	Precip
1	29	.15
2	32	.17
3	38	.20
7	78	.05
8	85	.04
9	83	.15

a.
```
a.  proc sql;
       select *
        from summer
     intersect all
     select *
        from winter;
```

b.
```
b.  proc sql;
     select *
        from summer
```

```
        outer union
        select *
           from winter;
```

c.
```
c.  proc sql;
        select *
           from summer
        union corr
        select *
           from winter;
```

d.
```
d.  proc sql;
        select *
           from summer
        union
        select *
           from winter;
```

4. Which PROC SQL step combines tables but does *not* overlay any columns? ?

a.
```
a.  proc sql;
    select *
           from groupa
        outer union
        select *
           from groupb;
```

b.
```
b.  proc sql;
        select *
           from groupa as a
        outer union corr
        select *
           from groupb as b;
```

c.
```
c.  proc sql;
        select coalesce(a.obs, b.obs)
               label='Obs', med, duration
           from groupa as a
           full join
           groupb as b
           on a.obs=b.obs;
```

d.
```
d.  proc sql;
        select *
           from groupa as a
        intersect
        select *
           from groupb as b;
```

5. Which statement is false regarding the keyword CORRESPONDING? ?

a. It cannot be used with the keyword ALL.

b. It overlays columns by name, not by position.

c. When used in EXCEPT, INTERSECT, and UNION set operations, it removes any columns not found in both tables.

d. When used in OUTER UNION set operations, it causes same-named columns to be overlaid.

6. Which PROC SQL step generates the following output from the tables *Dogs* and *pets*? ?

Dogs

Name	Price
FIFI	$101
GEORGE	$75
SPARKY	$136
TRUFFLE	$250

Pets

Name	Price	Arr
ANA	$25	09JAN2002
FIFI	$101	14MAR2002
GAO	$57	08DEC2001
GAO	$57	08DEC2001
SPARKY	$136	16SEP2002
TRUFFLE	$250	20DEC2002
ZEUS	$500	08JUN2002

Name	Price
ANA	$25
GAO	$57
ZEUS	$500

a. a.
```
proc sql;
 select name, price
    from pets
 except all
 select *
    from dogs;
```

b. b.
```
proc sql;
 select name, price
    from pets
 except
 select *
    from dogs;
```

c. c.
```
proc sql;
 select name, price
    from pets
 except corr all
 select *
    from dogs;
```

d. d.
```
proc sql;
 select *
    from dogs
 except corr
 select name, price
    from pets;
```

7. The *PROG1* and *PROG2* tables list students who took the PROG1 and PROG2 courses, respectively. Which PROC SQL step will give you the names of the students who took only the PROG1 class? ?

PROG1

FName	LName
Pete	Henry
Mary	Johnson
Alex	Kinsley
Dori	O'Neil

PROG2

FName	LName
Clara	Addams
Pete	Henry
Dori	O'Neil
Cindy	Phillips
Mandi	Young

PROG1 Only

FName	LName
Alex	Kinsley
Mary	Johnson

a. a.
```
proc sql;
 select fname, lname
    from prog1
 intersect
 select fname, lname
    from prog2;
```

b. b.
```
proc sql;
 select fname, lname
    from prog1
 except all
 select fname, lname
    from prog2;
```

c. c.
```
proc sql;
 select *
    from prog2
 intersect corr
 select *
    from prog1;
```

d. d.
```
proc sql;
 select *
    from prog2
 union
 select *
    from prog1;
```

8. Which PROC SQL step will return the names of all the students who took PROG1, PROG2, or both classes? ?

PROG1

FName	LName
Pete	Henry
Mary	Johnson
Alex	Kinsley
Dori	O'Neil

PROG2

FName	LName
Clara	Addams
Pete	Henry
Dori	O'Neil
Cindy	Phillips
Mandi	Young

PROG1, PROG2, or Both

FName	LName
Alex	Kinsley
Cindy	Phillips
Clara	Addams
Dori	O'Neil
Mandi	Young
Mary	Johnson
Pete	Henry

a. a.
```
proc sql;
 select fname, lname
    from prog1
 intersect
 select fname, lname
    from prog2;
```

b. b.
```
proc sql;
 select fname, lname
    from prog1
 outer union corr
 select fname, lname
    from prog2;
```

c. c.
```
proc sql;
 select fname, lname
    from prog1
 union
 select fname, lname
    from prog2;
```

d. d.
```
proc sql;
 select fname, lname
    from prog1
 except corr
 select fname, lname
    from prog2;
```

9. Which PROC SQL step will return the names of all the students who took both the PROG1 and PROG2 classes? ?

PROG1

FName	LName
Pete	Henry
Mary	Johnson
Alex	Kinsley
Dori	O'Neil

PROG2

FName	LName
Clara	Addams
Pete	Henry
Dori	O'Neil
Cindy	Phillips
Mandi	Young

PROG1 & PROG2

FName	LName
Dori	O'Neil
Pete	Henry

a. a.
```
proc sql;
 select fname, lname
    from prog1
 union
 select fname, lname
    from prog2;
```

b. b.
```
proc sql;
 select fname, lname
    from prog1
 except corr
 select fname, lname
    from prog2;
```

c. c.
```
proc sql;
 select fname, lname
    from prog1
 intersect all
 select fname, lname
    from prog2;
```

d. d.
```
proc sql;
 select fname, lname
    from prog1
 union corr
 select fname, lname
    from prog2;
```

10. Which PROC SQL step will generate the same results as the following DATA step? ?

```
data allstudents;
   set prog1 prog2;
   by lname;
run;
proc print noobs;
run;
```

PROG1

FName	LName
Pete	Henry
Mary	Johnson
Alex	Kinsley
Dori	O'Neil

PROG2

FName	LName
Clara	Addams
Pete	Henry
Dori	O'Neil
Cindy	Phillips
Mandi	Young

---

a. a.
```
proc sql;
 select fname, lname
    from prog1
 outer union corr
 select fname, lname
    from prog2
 order by lname;
```

b. b.
```
proc sql;
 select fname, lname
    from prog1
 union
 select fname, lname
    from prog2
 order by lname;
```

c. c.
```
proc sql;
 select fname, lname
    from prog2
 outer union
 select fname, lname
    from prog1
 order by lname;
```

d. d.
```
proc sql;
 select fname, lname
    from prog2
 union corr
 select fname, lname
    from prog1
 order by lname;
```

Answers

1. Correct answer: c

In set operations that use the operator EXCEPT, INTERSECT, or UNION, and no keyword, columns are overlaid based on their position in the SELECT clause. It does not matter whether the overlaid columns have the same name. When columns are overlaid, the column name is taken from the first table that is specified in the SELECT clause.

2. Correct answer: d

By default, when processing a set operation that contains the EXCEPT, INTERSECT, and UNION set operators, PROC SQL makes an extra pass through the data to eliminate duplicate rows. The keyword ALL is used to suppress that additional pass through the tables, allowing duplicate rows to appear in the result set. Because the OUTER UNION set operator displays *all* rows, the keyword ALL is invalid and *cannot* be used with OUTER UNION.

3. Correct answer: d

The output shown above contains all rows that are *unique* in the combined set of rows from both tables, and the columns have been *overlaid byposition*. This output is generated by a set operation that uses the set operator UNION without keywords.

4. Correct answer: a

The PROC SQL set operation that uses the set operator OUTER UNION without a keyword is the only code shown that does *not* overlay any columns in output.

5. Correct answer: a

The keyword CORRESPONDING (CORR) can be used alone or together with the keyword ALL.

6. Correct answer: b

This PROC SQL output includes all rows from the table *Pets* that do *not* appear in the table *Dogs. No* duplicates are displayed. A PROC SQL set operation that contains the set operator EXCEPT without keywords produces these results.

7. Correct answer: b

The set operator EXCEPT returns all the rows in the first table that do *not* appear in the second table. The keyword ALL suppresses the extra pass that PROC SQL makes through the data to eliminate duplicate rows. The EXCEPT operator when used alone will also produce the output specified in the question.

8. Correct answer: c

The set operator UNION returns all rows that are *unique* in the combined set of rows from both tables.

9. Correct answer: c

The set operator INTERSECT returns all rows that are common to both tables. Specifying the keyword ALL suppresses PROC SQL's additional pass through the data to eliminate duplicate rows.

10. Correct answer: a

The DATA step returns all rows from the first table along with all rows from the second table, maintaining the order specified in the BY statement. Same-named columns are overlaid by default. The set operator OUTER UNION returns all rows from both tables. The CORR keyword causes same-named columns to be overlaid. The ORDER BY clause causes the result rows to be ordered by values of the specified column **`(LName).`**